Dynamic DAG Scheduling on Multiprocessor Systems: Reliability, Energy and Makespan

Jing Huang, Renfa Li, Xun Jiao, Yu Jiang, Wanli Chang

***Abstract*—Multiprocessor systems are increasingly deployed in real-time applications, where reliability, energy consumption, and makespan are often the main scheduling objectives. In this work, we investigate dynamic scheduling of tasks modelled by directed acyclic graphs (DAGs), which is an NP-hard problem with all existing methods being heuristics. Our contributions have two steps: (i) Assuming that the allocation of DAG nodes to processors is given, we propose OEA (Optimal Energy Allocation) and SOEA (Search-based OEA) — the first optimal methods that minimise the energy consumption whilst satisfying the reliability requirement — for homogeneous and heterogeneous systems, respectively; (ii) We present a novel scheduling algorithm ODS (Out-Degree Scheduling) that allocates the DAG nodes according to their out-degrees, and considering energy consumption, reliability, as well as dynamic finish time. ODS dominates the widely applied HEFT (Heterogeneous Earliest Finish Time) in makespan. Combining SOEA with ODS makes a complete solution to the problem of dynamic DAG scheduling on multiprocessor systems, and achieves generally better results compared to the existing approaches. Specifically, in most cases, we are better on all the three objectives, i.e., reliability, energy as well as makespan, and in other cases we are better on some of the objectives.**

***Index Terms*—DAG, dynamic scheduling, multiprocessor systems, reliability, energy consumption, makespan**

I. Introduction

The complex functionalities of the emerging real-time applications, such as in the automotive, industrial automation, and robotics domains, require multiprocessor systems for implementation. The tasks need to be modelled as directed acyclic graphs (DAGs) to capture the dependency. Many of these applications are safety-critical with constraints on reliability. That is, the probability of failure must be very low during runtime.

Another important objective is to minimise the energy consumption. Taking the autonomous vehicles as an example, the power consumption of the computing system is similar to that of traction [1], [2]. The energy storage of battery-powered devices, such as electric vehicles and drones, is especially limited.

For the real-time applications in practice, there is a trend that tasks may get added or removed online with no predictable pattern. This makes static scheduling difficult and motivates dynamic scheduling, where *makespan*, i.e., the time it takes to complete the execution of a DAG, is often the performance metric. An example is the move towards the AUTOSAR (AUTomotive Open System ARchitecture) Adaptive standard [3] in the automotive industry.

Main contributions: In this work, we study dynamic DAG scheduling on multiprocessor systems, considering reliability, energy consumption, and makespan. This is known to be an NP-hard problem, where all existing methods are heuristics [4]–[7]. Our contributions have two steps:

- Assuming that the allocation of DAG nodes to processors is given, we propose the first optimal methods that minimise the energy consumption whilst satisfying the reliability constraint. OEA (Optimal Energy Allocation) offers a closed-form solution for homogeneous architectures, and SOEA (Search-based OEA) is an algorithm built upon binary search for heterogeneous architectures.
- We report ODS (Out-Degree Scheduling), a novel scheduling algorithm that allocates the DAG nodes according to their out-degrees, and considering energy consumption, reliability, as well as dynamic finish time. ODS dominates the widely applied HEFT (Heterogeneous Earliest Finish Time) [8], [9] in the way that ODS achieves shorter makespan than HEFT in certain cases and is at least as good as it in all cases. Combining SOEA with ODS makes a complete solution to the problem of dynamic DAG scheduling on multiprocessor systems. Extensive evaluations on DAGs with various degrees of parallelism show that it achieves generally better results compared to the existing approaches. Specifically, in most cases, we are better on all the three objectives, i.e., reliability, energy as well as makespan, and in other cases we are better on some of the objectives. Both SOEA and ODS have polynomial time complexity.

To summarise, the studied problem is decomposed into two. One is allocation of nodes to processors, with all the three objectives considered, and the other is to compute the frequencies of the processors that minimise the energy consumption under the reliability constraint.

The rest of this paper is organised as follows. Section II reviews the literature considering reliability, energy consumption, and timing performance, separately and jointly, in embedded systems. Section III describes the models and problem

J. Huang and R. Li are with the College of Computer Science and Electronic Engineering, and the Key Laboratory for Embedded and Network Computing of Hunan Province, Hunan University, China (E-mail: jingh@hnu.edu.cn, lirenfa@hnu.edu.cn).

X. Jiao is with the Department of Electrical and Computer Engineering, Villanova University, USA (E-mail: xun.jiao@villanova.edu).

Y. Jiang is with the School of Software, Tsinghua University, China (E-mail: jy1989@mail.tsinghua.edu.cn).

W. Chang is with the Department of Computer Science, University of York, UK (E-mail: wanli.chang@york.ac.uk).

Manuscript received 17 April, 2020; revised 17 June, 2020; accepted 6 July, 2020. This article was presented in the International Conference on Embedded Software 2020 and appears as part of the ESWEEK-TCAD special issue.

W. Chang is the corresponding author.

formulation. Section IV presents the methods for optimal energy consumption. The new scheduling algorithm ODS is given in Section V. The experimental results are reported in Section VI and Section VII makes the concluding remarks.

II. Related Work

There has been a large body of works investigating and trying to optimise reliability, energy consumption, as well as timing performance, in embedded systems [10], [6], [11], [12], [4]. On reliability, the target is to reduce the occurrence of faults, transient or permanent, during task execution. These faults may be caused by several factors, including hardware failure, frequent temperature variation, and high temperature [7]. Transient faults, which are related to soft-error reliability, are much more likely to occur in practice than permanent faults, which are related to lifetime reliability [13], [14]. Therefore, many works focus on transient faults. A widely accepted reliability model, proposed in [15], approximates the possibility of no transient fault during the time interval t with an exponential distribution $e^{-\lambda t}$, which indicates that reliability decreases with time. Therefore, besides redundancy [16], a common approach to improve reliability is decreasing the task execution time [17], [18].

Energy consumption is often reduced with DVFS (Dynamic Voltage and Frequency Scaling), which scales processors' voltage and frequency simultaneously as a middleware implemented on the operating system level [19]. Since high frequency is the main cause of high energy consumption for a processor, proper use of DVFS effectively saves energy [20].

Many works considering timing performance aim to complete the execution of tasks as soon as possible, i.e., shorten the *makespan* or *scheduling length*. A task's makespan is mainly determined by the scheduling algorithm. HEFT [8] is the most popular DAG scheduling algorithm and widely believed to achieve the shortest makespan in many cases. There are other methods that take energy cost and reliability into account, such as LEC (Least Energy Cost) [6] and MR (Maximum Reliability) [21].

There have also been works that jointly consider reliability, energy consumption, and timing performance, in embedded systems. Reliability is maximised under hard energy constraints on real-time applications in [4], which can be applied only to uniprocessor systems. A task allocation approach LRDSA (Local Reliabiliy-Driven Scheduling Algorithm) is reported in [5], enabling a trade-off between performance and reliability. ESRG (energy-efficient scheduling with reliability goal) is presented in [6] to improve energy efficiency and reliability simultaneously. An evolutionary algorithm is employed in [7], which is computationally very heavy and does not suit dynamic scheduling. These state-of-the-art algorithms, including HEFT, LEC, MR, LRDSA, and ESRG, consider similar models, platforms, and objectives to us. We will compare our complete solution to dynamic DAG scheduling on multiprocessor systems with them.

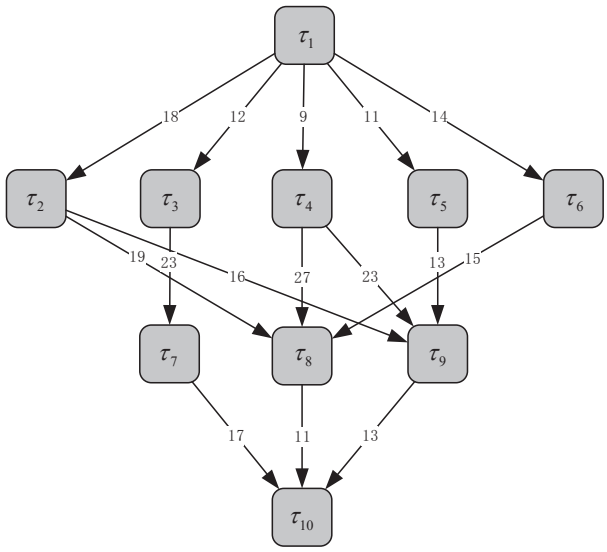


Fig. 1. An example DAG task 10 sub-tasks

III. Models and Problem Formulation

A. Application Model

A DAG task model is represented by $G = (N, E)$, where N is a set of nodes and E is a set of edges. Each node $\tau_i \in N$ denotes a sub-task of the DAG, and each directed edge $e_{i,j} \in E$ specifies an execution order that the sub-task τ_j can only start after the sub-task τ_i is completed. Correspondingly, τ_j is called as an immediate successor of τ_i, and τ_i is an immediate predecessor of τ_j. We use $succ(\tau_i)$ and $pre(\tau_i)$ to denote the set of immediate successors and predecessors of τ_i, respectively. In addition, each edge $e_{i,j}$ has a weight $w_{i,j}$ that represents the communication cost between the sub-tasks τ_i and τ_j. If two sub-tasks are allocated to the same processor, the communication cost between them is commonly ignored.

An example DAG task with 10 sub-tasks $\{\tau_1, \tau_2, \ldots, \tau_{10}\}$ is shown in Figure 1, where $\tau_2, \tau_3, \tau_4, \tau_5, \tau_6$ can only start after τ_1 is completed, and be executed in parallel. Table I presents the computation matrix of this example on a three-processor system, which lists the execution time of each sub-task on every processor at the maximum frequency f_{max}. On a heterogeneous platform, different processors offer different computational capabilities [22]. The processor parameters will be explained in the next sub-sections on the energy and reliability model. This DAG task in Figure 1 with the computation matrix and processor parameters has been widely used in the literature. It will also be used as an illustrative example in this work, and be referred to in the later sections.

B. Energy Model

The power dissipation of a processor mainly consists of frequency-dependent dynamic consumption, frequency-independent dynamic consumption, and static consumption.

TABLE I
THE COMPUTATION MATRIX OF THE DAG TASK IN FIGURE 1 ON A THREE-PROCESSOR SYSTEM WITH PARAMETERS

	The computation matrix										Processor parameters				
	T_{τ_1}	T_{τ_2}	T_{τ_3}	T_{τ_4}	T_{τ_5}	T_{τ_6}	T_{τ_7}	T_{τ_8}	T_{τ_9}	$T_{\tau_{10}}$	P^*	α	c	d	λ_F
u_1	14	13	11	13	12	13	7	5	18	21	0.04	2.9	0.8	3.0	2.0E-4
u_2	16	19	13	8	13	16	15	11	12	7	0.04	2.9	0.9	3.0	1.3E-4
u_3	9	18	19	17	10	9	11	14	20	16	0.04	3.0	1.0	3.0	0.9E-4

Among them, frequency-dependent dynamic power consumption is the dominant component, and can be expressed by,

$$P = \xi c v^2 f, \tag{1}$$

where ξ is an activity factor, c is the loading capacitance, v is the supply voltage, and f is the clock frequency. Given that $f \propto v$, we have $P \propto cf^{\alpha}$, where α is approximately 3. For ease of discussion, we model the frequency-dependent dynamic power consumption of a processor as cf^{α}, and use P^* to denote the frequency-independent dynamic power consumption and static power consumption. The overall power consumption of a processor u_k is then,

$$P = P^* + cf^{\alpha}. \tag{2}$$

For convenience, we normalise the frequency of a processor such that $f_{max} = 1$. Assuming T_{τ_i} to be the execution time of the sub-task τ_i on a processor with its maximum frequency f_{max}, when the operating frequency is f_i ($f_i \leq f_{max}$), the execution time can be calculated as,

$$t_{\tau_i} = T_{\tau_i} \times \frac{f_{max}}{f_i} = T_{\tau_i} \times \frac{1}{f_i}. \tag{3}$$

The energy required to complete τ_i is the product of the processor power consumption and the execution time,

$$E_{\tau_i}(f_i) = P \times t_{\tau_i} = (P^* + cf_i^{\alpha}) \times \frac{T_{\tau_i}}{f_i}. \tag{4}$$

The energy consumption of a DAG task equals the sum of all sub-tasks,

$$E_G(\boldsymbol{f}) = E_{\tau_1}(f_1) + E_{\tau_2}(f_2) + \cdots + E_{\tau_n}(f_n), \tag{5}$$

where $\boldsymbol{f} = (f_1, f_2, \ldots, f_n)$ represents a vector.

C. Reliability Model

The reliability of a task is defined as the probability of no fault during its execution. As discussed in Section II, similar to many other works, we focus on the dominant transient faults, which are related to processor frequency and can be modelled with the following exponential distribution,

$$\lambda(f) = \lambda_F \times 10^{\frac{d(1-f)}{1-f_{\min}}}, \tag{6}$$

where λ_F stands for the average number of faults per second at the maximum frequency, d is a hardware-related constant, and f_{min} is the minimum available frequency.

The longer it takes the task to execute, the higher the probability of faults. Based on (6), the reliability of a sub-task τ_i executed on a processor with frequency f_i can be calculated as,

$$R_{\tau_i}(f_i) = e^{-\lambda(f_i) \times \frac{T_{\tau_i}}{f_i}}. \tag{7}$$

A reliable DAG task requires that all the sub-tasks are successfully executed without faults. Therefore, the reliability $R(G)$ of a DAG task is equal to the product of all sub-tasks,

$$R_G(\boldsymbol{f}) = R_{\tau_1}(f_1) \times R_{\tau_2}(f_2) \times \cdots \times R_{\tau_n}(f_n). \tag{8}$$

D. Problem Formulation

With the above models, our problem can be formulated as follows: Given a DAG task $G = (N, E)$, a set of DVFS-enabled processors $U = \{u_1, u_2, \ldots, u_m\}$, and a reliability requirement R_{req}, we aim to find a scheduling algorithm that minimises the makespan MS_G, which is counted from the DAG arrival to its execution completion, and the energy consumption E_G,

$$Minimise\ MS_G\ and\ E_G(\boldsymbol{f}), \tag{9}$$

subject to

$$R_G(\boldsymbol{f}) = R_{\tau_1}(f_1) \times R_{\tau_2}(f_2) \times \cdots \times R_{\tau_n}(f_n) \geq R_{req}.$$

Similarly, we can also optimise MS_G and the reliability R_G under the energy constraint $E_G \leq E_{req}$, which is particularly useful for the battery-powered devices, such as electric vehicles and drones, when their energy storage is mostly drained out.

IV. OPTIMAL ENERGY ALLOCATION

In this section, we assume that the allocation of DAG nodes, i.e., the sub-tasks, to processors is known (which will be done by ODS in the next section), and present the first optimal methods — OEA and SOEA — that minimise the energy consumption with the given reliability requirement. The makespan will be dealt with in node allocation by ODS.

Excluding makespan, our problem becomes,

$$\begin{aligned} &Minimise\ E_G(\boldsymbol{f}) \\ &\qquad = E_{\tau_1}(f_1) + E_{\tau_2}(f_2) + \cdots + E_{\tau_n}(f_n), \end{aligned} \tag{10}$$

subject to

$$R_G(\boldsymbol{f}) = R_{\tau_1}(f_1) \times R_{\tau_2}(f_2) \times \cdots \times R_{\tau_n}(f_n) \geq R_{req}.$$

This is a multi-variable optimisation problem. We use the KKT (Karush-Kuhn-Tucker) method to solve it and construct

$$L(\boldsymbol{f}, o) = E_G(\boldsymbol{f}) + o\left(R_{req} - R_G(\boldsymbol{f})\right), \tag{11}$$

which can be differentiated to be,

$$\frac{\partial L(\boldsymbol{f}, o)}{\partial f_i} = \frac{\partial E_{\tau_i}(f_i)}{\partial f_i} - o\frac{\partial R_G(\boldsymbol{f})}{\partial f_i}, \tag{12}$$

where

$$\frac{\partial E_{\tau_i}(f_i)}{\partial f_i} = T_{\tau_i}\left(c(\alpha-1)f_i^{\alpha-2} - \frac{P^*}{f_i^2}\right), \tag{13}$$

and

$$\frac{\partial R_G(\boldsymbol{f})}{\partial f_i} = R_{\tau_1}(f_1)\times\cdots\times\frac{\partial R_{\tau_i}(f_i)}{\partial f_i}\times\cdots\times R_{\tau_n}(f_n). \tag{14}$$

According to (7), we get

$$\begin{aligned}
&\frac{\partial R_{\tau_i}(f_i)}{\partial f_i}\\
&= -e^{-\lambda(f_i)\times\frac{T_{\tau_i}}{f_i}}\times\left(\lambda'(f_i)\frac{T_{\tau_i}}{f_i} + \left(\frac{T_{\tau_i}}{f_i}\right)'\lambda(f_i)\right)\\
&= -e^{-\lambda(f_i)\times\frac{T_{\tau_i}}{f_i}}\times\left(\lambda'(f_i)\frac{T_{\tau_i}}{f_i} - \frac{T_{\tau_i}}{{f_i}^2}\lambda(f_i)\right)\\
&= e^{-\lambda(f_i)\times\frac{T_{\tau_i}}{f_i}}\\
&\quad\times\frac{T_{\tau_i}}{f_i}\left(\lambda_F\times 10^{\frac{d(1-f_i)}{1-f_{\min}}}\times\frac{d\ln 10}{1-f_{\min}} + \frac{\lambda(f_i)}{f_i}\right)\\
&= e^{-\lambda(f_i)\times\frac{T_{\tau_i}}{f_i}}\times\frac{T_{\tau_i}}{f_i}\left(\lambda(f_i)\times\frac{d\ln 10}{1-f_{\min}} + \frac{\lambda(f_i)}{f_i}\right)\\
&= R_{\tau_i}(f_i)\times\lambda(f_i)\times\frac{T_{\tau_i}}{f_i}\left(\frac{d\ln 10}{1-f_{\min}} + \frac{1}{f_i}\right).
\end{aligned} \tag{15}$$

Taking (15) into (14), we have

$$\begin{aligned}
\frac{\partial R_G(\boldsymbol{f})}{\partial f_i} &= R_{\tau_1}(f_1)\times\cdots\times R_{\tau_i}(f_i)\times\cdots\times R_{\tau_n}(f_n)\\
&\quad\times\lambda(f_i)\times\frac{T_{\tau_i}}{f_i}\left(\frac{d\ln 10}{1-f_{\min}} + \frac{1}{f_i}\right)\\
&= R_G(\boldsymbol{f})\times\lambda(f_i)\times\frac{T_{\tau_i}}{f_i}\left(\frac{d\ln 10}{1-f_{\min}} + \frac{1}{f_i}\right).
\end{aligned} \tag{16}$$

Equating (12) to 0,

$$\begin{aligned}
&T_{\tau_i}\left(c(\alpha-1)f_i^{\alpha-2} - \frac{P}{f_i^2}\right) =\\
&oR_G(\boldsymbol{f})\times\lambda(f_i)\times\frac{T_{\tau_i}}{f_i}\left(\frac{d\ln 10}{1-f_{\min}} + \frac{1}{f_i}\right).
\end{aligned} \tag{17}$$

Taking $R_G(\boldsymbol{f}) = R_{req}$ into (17), we can obtain,

$$\begin{aligned}
&c(\alpha-1)f_i^{\alpha-1} - \frac{P^*}{f_i} =\\
&oR_{req}\times\lambda(f_i)\left(\frac{d\ln 10}{1-f_{\min}} + \frac{1}{f_i}\right).
\end{aligned} \tag{18}$$

A. OEA for Homogeneous Systems

Based on (18), for homogeneous systems, where all processors are identical, the energy consumption is minimal with equal frequencies, and we set,

$$f = f_1 = f_2 = \cdots = f_n.$$

According to (7) and (8),

$$\begin{aligned}
&R_G(f)\\
&= R_{\tau_1}(f)\times R_{\tau_1}(f)\times\cdots\times R_{\tau_1}(f)\\
&= e^{-\lambda(f)\times\frac{T_{\tau_1}}{f}}\times e^{-\lambda(f)\times\frac{T_{\tau_2}}{f}}\times\cdots\times e^{-\lambda(f)\times\frac{T_{\tau_n}}{f}}\\
&= e^{-\left(T_{\tau_1}+T_{\tau_2}+\cdots+T_{\tau_n}\right)\times\frac{\lambda(f)}{f}}.
\end{aligned} \tag{19}$$

We let $R_G(\boldsymbol{f})$ take the lower bound R_{req},

$$\ln R_{req} = -\left(T_{\tau_1}+T_{\tau_2}+\cdots T_{\tau_n}\right)\times\frac{\lambda(f)}{f}, \tag{20}$$

i.e.,

$$\frac{-\ln R_{req}}{T_{\tau_1}+T_{\tau_2}+\cdots+T_{\tau_n}} = \frac{\lambda(f)}{f}. \tag{21}$$

The variable f in (21) is difficult to solve directly. Observing (21), we can find that $\lambda(f)$ is a monotonically decreasing function of f, and so is $\frac{\lambda(f)}{f}$. Since the left hand side of (21) is a constant, the solution can be iteratively approached.

B. SOEA for Heterogeneous Systems

For heterogeneous systems with different processors, the frequencies that optimise the energy consumption may be different. We denote the parameters of the processor k as α_k, P_k^*, c_k, $\lambda_k(f)$, and $\lambda_{k,F}$. Correspondingly, (18) can be rewritten as,

$$\begin{aligned}
&c_k(\alpha_k-1)f_i^{\alpha_k-1} - \frac{P_k^*}{f_i} =\\
&oR_{req}\times\lambda(f_i)\left(\frac{d_k\ln 10}{1-f_{k,\min}} + \frac{1}{f_i}\right),
\end{aligned} \tag{22}$$

where the indices k and i indicate that the task τ_i is allocated to the processor k. There are two variables in (22), the frequency f_i and Lagrange factor o. To solve for f_i, the Lagrange factor o needs to be determined.

We first analyse the relationship between o and f_i. To this end, we treat the Lagrange factor o as a function of f_i,

$$o(f_i) = \frac{c_k(\alpha_k-1)f_i^{\alpha_k-1} - \frac{P_k^*}{f_i}}{R_{req}\times\lambda(f_i)\left(\frac{d_k\ln 10}{1-f_{k,\min}} + \frac{1}{f_i}\right)}, \tag{23}$$

Considering the numerator of (23),

$$y(f_i) = c_k(\alpha_k-1)f_i^{\alpha_k-1} - \frac{P_k^*}{f_i}, \tag{24}$$

we have

$$y'(f_i) = c_k(\alpha_k-1)^2 f_i^{\alpha_k-2} + \frac{P_k^*}{{f_i}^2} > 0, \tag{25}$$

which means that $y(f_i)$ is a monotonically increasing function of f_i. On the other hand, the denominator of (23)

$$R_{req}\times\lambda(f_i)\left(\frac{d_k\ln 10}{1-f_{k,\min}} + \frac{1}{f_i}\right) \tag{26}$$

monotonically decreases with f_i. Therefore, $o(f_i)$ is a monotonically increasing function of f_i. From (7), we can get that $R_{\tau_i}(f_i)$ is a monotonically increasing function of f_i. Therefore, $R_G(\boldsymbol{f}) = \prod_{i=1}^{n} R_{\tau_i}(f_i)$ monotonically increases with f_i as well as $o(f_i)$.

With the above analysis, we can apply a simple binary search to find the o that corresponds to the set of frequencies satisfying $R_G(\boldsymbol{f}) = R_{req}$. Note that as the Lagrange factor,

Algorithm 1 Search-based Optimal Energy Allocation

Input: $G = (N, E), U, R_{req}$.
Output: $f_1, f_2, \ldots, f_n$.
1: Compute ub and lb according to (27); //ub and lb are the upper and lower bounds of the search region, respectively.
2: **while** $(ub - lb > \varepsilon)$ **do**
3: $mid \leftarrow (ub + lb)/2$;
4: **for** each processor k **do**
5: $f_{lb} \leftarrow f_{k,min}$, $f_{ub} \leftarrow f_{k,max}$, $f \leftarrow (f_{lb} + f_{ub})/2$;
6: **while** $(f_{ub} - f_{lb} > \epsilon)$ **do**
7: **if** $o(f) < mid$ **then**
8: $f_{lb} \leftarrow f$;
9: **else**
10: $f_{ub} \leftarrow f$;
11: **end if**
12: $f \leftarrow (f_{lb} + f_{ub})/2$;
13: **end while**
14: **for** each task τ_i allocated to the processor u_k **do**
15: $f_i \leftarrow f$;
16: **end for**
17: **end for**
18: **if** $R_G(\boldsymbol{f}) - R_{req} < 0$ **then**
19: $lb \leftarrow mid$;
20: **else**
21: $ub \leftarrow mid$;
22: **end if**
23: **end while**
24: **return** $f_1, f_2, \ldots, f_n$;

o is the same for all frequencies f_i $(1 \leq i \leq n)$. The search range of o is,

$$\min_{1 \leq k \leq m} o(f_{k,min}) \leq o \leq \max_{1 \leq k \leq m} o(f_{k,max}), \tag{27}$$

where $o(f_{k,min})$ and $o(f_{k,max})$ are the minimum and maximum frequency of the processor k, respectively.

The problem studied in this work is decomposed in the way that one stage, i.e., computing the frequencies of processors that minimise the energy consumption with the given reliability requirement, after the allocation of nodes to processors, can be solved with an optimal approach (the first in this context). Once the problem is formulated, applying the KKT method is quite standard, after which, a series of analyses (customised in this problem setting) on the monotonic relationships are required to reach the final solution.

Energy constraint: We can also maximise the reliability and respect a constraint on energy consumption. Following similar steps, (17) can be obtained. Treating $oR_G(\boldsymbol{f})$ together,

$$y(f_i) = oR_G(\boldsymbol{f}) = \frac{c_k\left(\alpha_k - 1\right) f_i^{\alpha_k - 1} - \dfrac{P_k^*}{f_i}}{\lambda\left(f_i\right)\left(\dfrac{d_k \ln^{10}}{1 - f_{k,\min}} + \dfrac{1}{f_i}\right)}, \tag{28}$$

which is in a similar form to (23). Therefore, $y(f_i)$ is a monotonically increasing function of f_i. From (4) and (5), we can get that $E_G(\boldsymbol{f})$ monotonically increases with f_i. A simple binary search can be deployed to find the y that corresponds to the set of frequencies satisfying $E_G(\boldsymbol{f}) = E_{req}$. Since $0 \leq R_G(\boldsymbol{f}) \leq 1$, the search range of y is the same as (27) on the upper bound and 0 on the lower bound.

The search algorithm: As discussed above, a binary search algorithm is able to return the optimal solution to both the problems with reliability and energy consumption as a constraint, respectively. We present SOEA in Algorithm 1, using the energy optimisation as an example for illustration. The search range is computed in Line 1. The frequencies corresponding to $o = mid$ are derived in Lines 4-17. The bounds of the binary search are updated in Lines 18-22. The time complexity of SOEA is,

$$O\left(log(L_o/\varepsilon) \times m \times log(L_f/\varepsilon) \times n\right),$$

where L_o is the search range from (27), m is the number of processors, n is the number of tasks,

$$L_f = \max_{1 \leq k \leq m} \{f_{k,max} - f_{k,min}\},$$

and ε is the accuracy, which in our work is set to 10^{-5}. Whilst SOEA is motivated in the DAG setting, it can be generally applied as long as the allocation of tasks to processors is known.

V. Out-Degree Scheduling

In the last section, we present the methods that minimise the energy consumption or maximise the reliability, assuming that the allocation of DAG nodes (sub-tasks) to processors is known. In this section, we will report how to perform such allocation.

In task scheduling on multiprocessor systems, besides processor allocation, the execution order of tasks also needs to be determined, for which there exist many approaches. In this work, we take a widely applied method, which executes tasks in the decreasing order of their up-rank-values urv,

$$urv(\tau_i) = \frac{1}{m}\sum_{k=1}^{m} T_{k,\tau_i} + \max_{\tau_j \in succ(\tau_i)} \{w_{ij} + urv(\tau_j)\}, \tag{29}$$

where m is the number of processors as defined earlier in the paper. Essentially, in a DAG, a node with long execution time and a heavy successor (in terms of both communication cost and execution time) is prioritised to be executed. The number of successors is not considered. For the example in Figure 1 and Table I, we compute the up-rank-values of the sub-tasks $\tau_1, \tau_2, \ldots, \tau_{10}$ to be 108, 77, 80, 80, 69, 63, 43, 36, 44, 14. Therefore, the execution order is $\tau_1, \tau_3, \tau_4, \tau_2, \tau_5, \tau_6, \tau_9, \tau_7, \tau_8, \tau_{10}$.

Following the execution order, the tasks get allocated to the processors. The existing task allocation methods for DAG in the literature mainly aim to shorten the makespan. The most popular one is HEFT, which is believed to achieve the shortest makespan in many cases. There are other methods that take energy cost and reliability into account, such as LEC and MR. Table II reports the scheduling results on the example in Figure 1 and Table I by HEFT, LEC, and MR, where st_{τ_i} and ft_{τ_i} denote the start and finish time of the sub-task τ_i, respectively.

TABLE II
SCHEDULING RESULTS ON THE EXAMPLE IN FIGURE 1 AND TABLE I BY HEFT, LEC, AND MR

	HEFT						LEC						MR					
τ_i	u_k	f_i	st_{τ_i}	ft_{τ_i}	E_{τ_i}	R_{τ_i}	u_k	f_i	st_{τ_i}	ft_{τ_i}	E_{τ_i}	R_{τ_i}	u_k	f_i	st_{τ_i}	ft_{τ_i}	E_{τ_i}	R_{τ_i}
1	3	1	0	9	9.36	0.9992	3	1	0	9	9.36	0.9992	3	1	0	9	9.36	0.9992
2	1	1	27	40	10.92	0.9974	1	1	32	45	10.92	0.9974	3	1	9	27	18.72	0.9984
3	3	1	9	28	19.76	0.9983	1	1	21	32	9.24	0.9978	2	1	21	34	12.22	0.9983
4	2	1	18	26	7.52	0.9990	2	1	18	26	7.52	0.9990	2	1	34	42	7.52	0.9990
5	3	1	28	38	10.4	0.9991	1	1	45	57	10.08	0.9976	3	1	27	37	10.4	0.9991
6	2	1	26	42	15.04	0.9979	3	1	9	18	9.36	0.9992	3	1	37	46	9.36	0.9992
7	3	1	38	49	11.44	0.9990	1	1	57	64	5.88	0.9986	3	1	57	68	11.44	0.9990
8	1	1	57	62	4.2	0.9990	1	1	64	69	4.2	0.9990	1	1	69	74	4.2	0.9990
9	2	1	56	68	11.28	0.9984	2	1	70	82	11.28	0.9984	2	1	50	62	11.28	0.9984
10	2	1	73	80	6.58	0.9991	2	1	82	89	6.58	0.9991	2	1	85	92	6.58	0.9991
	$E_G = 106.5, MS_G = 80, R_G = 0.9865$						$E_G = 84.420, MS_G = 89, R_G = 0.9854$						$E_G = 101.080, MS_G = 92, R_G = 0.9887$					

TABLE III
SCHEDULING RESULTS ON THE EXAMPLE IN FIGURE 1 AND TABLE I BY ODS

	ODS					
τ_i	u_k	f_i	st_{τ_i}	ft_{τ_i}	E_{τ_i}	R_{τ_i}
1	3	1	0	9	9.36	0.9992
2	3	1	9	27	18.72	0.9984
3	1	1	21	32	9.24	0.9978
4	2	1	18	26	7.52	0.9990
5	1	1	32	44	10.08	0.9976
6	3	1	27	36	9.36	0.9992
7	1	1	44	51	5.88	0.9986
8	1	1	53	58	4.2	0.9990
9	2	1	57	69	11.28	0.9984
10	2	1	69	76	6.58	0.9991
	$E_G = 92.2, MS_G = 76, R_G = 0.9863$					

It can be seen that for HEFT,

$$E_G = 106.5, MS_G = 80, R_G = 0.9865,$$

for LEC,

$$E_G = 84.420, MS_G = 89, R_G = 0.9854,$$

and for MR,

$$E_G = 101.080, MS_G = 92, R_G = 0.9887,$$

which illustrates the different advantages of these three methods. That is, HEFT obtains the shortest makespan $MS_G = 80$, LEC gets the least energy cost $E_G = 84.420$, and MR achieves the highest reliability $R_G = 0.9887$. As discussed earlier, our SOEA can be used in combination with any of these three algorithms.

Below we describe the proposed method that allocates DAG sub-tasks (nodes) to processors, considering the dynamic finish time of sub-tasks, energy consumption of processors, and reliability of sub-tasks on processors. The complexity of DAG scheduling mainly comes from the dependency between nodes. Generally, executing a node with many successors earlier is in favour of short makespan for the entire DAG. For instance, in Figure 1, the sub-task τ_1 has an out-degree of 5 with 5 immediate successors $\tau_2, \tau_3, \tau_4, \tau_5, \tau_6$, where the out-degree of a node is defined as the number of its immediate successors. Prioritising τ_1 in allocation potentially benefits all its successors. Therefore, we only allocate those nodes with larger out-degrees to processors with early finish time. That is, we would like to execute those sub-tasks with larger number of successors earlier. This is different from HEFT, where all the nodes try to go for the processors with early finish time.

In HEFT, following the up-rank-values and ignoring the out-degree, a node with a large number of light successors (short execution time and low communication cost) may be delayed as the system scheduling bottleneck, which harms the DAG makespan.

The major steps of ODS are outlined as follows and shown in Algorithm 2.

- Calculate the out-degree $OD(\tau_i)$ of each sub-task τ_i.
- Put all sub-tasks in a queue ODQ in the decreasing order of $OD(\tau_i)$. Ties are broken with the up-rank-value.
- If a sub-task τ_i is in the region $[ODQ[0], ODQ[l]]$, i.e., with a relatively larger out-degree, it will be allocated to the processor k with

$$\min_{1 \le k \le m} \{ft_{\tau_i,k} + \theta(1 - R_{\tau_i,k})T_{\tau_i,k}\}, \tag{30}$$

  where $ft_{\tau_i,k}$, $R_{\tau_i,k}$, and $T_{\tau_i,k}$ are the finish time, reliability, and execution time of the sub-task τ_i on the processor k, respectively, considering the maximum frequency [5].
- If a sub-task τ_i is in the region $[ODQ[l+1], ODQ[n-1]]$, i.e., with a relatively smaller out-degree, it will be allocated to the processor k with

$$\min_{1 \le k \le m} E_{\tau_i,k}, \tag{31}$$

  where $E_{\tau_i,k}$ is the energy consumption of the sub-task τ_i on the processor k, considering the maximum frequency.
- Calculate the up-rank-value $urv(\tau_i)$ for each sub-task τ_i.
- Allocate the sub-tasks in the decreasing order of $urv(\tau_i)$ to the processors based on (30) and (31).

The finish time $ft_{\tau_i,k}$ is equal to $st_{\tau_i,k} + T_{\tau_i,k}$, where $st_{\tau_i,k}$ is the start time of the sub-task τ_i on the processor k, and can be calculated by

$$\max\{ft_{\tau_j} + w_{ji} | \tau_j \in pre(\tau_i)\}.$$

The start time of the entry sub-task is 0. The parameter θ ($\theta \ge 0$) is used to balance the finish time with reliability, and the parameter l ($0 \le l \le n-1$) provides a trade-off between finish time and energy cost.

ODS can be iterated over l and θ to find the solution that suits the demand best. When $\theta = 0$ and $l = n - 1$,

Algorithm 2 Out-Degree Scheduling

Input: $G = (N, E), U, l, \theta$.
Output: E_G, MS_G, R_G.
1: **for** $i = 0$ to $n - 1$ **do**
2: Calculate the out-degree $OD(\tau_i)$ of the sub-task τ_i;
3: Calculate the up-rank-value $urv(\tau_i)$ of the sub-task τ_i;
4: **end for**
5: Push all sub-tasks into the queue ODQ[] in the decreasing order of $OD(\tau_i)$;
6: Push all sub-tasks into the queue rQ[] in the decreasing order of $urv(\tau_i)$;
7: **for** $j = 0$ to $n - 1$ **do**
8: $\tau_i \leftarrow rQ[j]$;
9: **if** $\tau_i \in [ODQ[0], ODQ[l]]$ **then**
10: Allocate τ_i to the processor satisfying (30);
11: **else**
12: Allocate τ_i to the processor satisfying (31);
13: **end if**
14: **end for**
15: **return** E_G, MS_G, R_G;

ODS is equivalent to HEFT. Therefore, ODS is able to obtain solutions that are at least as good as HEFT. Table III reports the scheduling results on the example in Figure 1 and Table I by ODS. The target is to find the shortest makespan, which is achieved when the parameter l is 3 and the parameter θ is 0. Comparing the results in Table II and Table III, ODS achieves a shorter makespan 76 than the 80 from HEFT, which has been widely believed to be the shortest makespan for this popular example in Figure 1 and Table I. ODS has polynomial time complexity $O(n^3 m L_\theta)$, where L_θ is the number of iterations over θ, and the number of iterations over l is equal to the number of sub-tasks n.

Novelty of ODS: There have been some works in the real-time systems community that use out-degrees of nodes when dealing with DAGs [23], [24]. However, they are on response-time analysis, instead of scheduling approaches. Out-degrees are taken in the high-performance computing community to determine the scheduling order of DAG nodes, such as [25]. By contrast, we perform allocation of nodes to processors (the scheduling order in each processor is implicit), which is essential for heterogeneous architectures widely deployed in embedded systems. In ODS, the out-degrees of nodes are leveraged as a judging threshold, not directly employed for allocation. That is, the nodes with larger out-degrees (likely to impact the makespan strongly) are allocated to the processor with the earliest finish time, and those with smaller out-degrees (unlikely to impact the makespan strongly) to the processor with the lowest energy consumption, both in the order of the up-rank-value, which is known to be an effective metric.

ODS is a framework where HEFT is one instance often dominated by other instances. There are two important concepts in ODS that are missing in HEFT. First, the out-degrees should be considered in the scheduling and allocation, even when only timing is of concern. Otherwise, as discussed earlier, contention over the processors with early finish time may be heavy for a node with a large number of light successors, which could become the system bottleneck and prolong the makespan. Second, ODS takes into account the energy consumption, which is completely ignored by HEFT. It makes little sense for those nodes that weakly impact the makespan to still go for processors with early finish time (they will not get the early processors anyway). Instead, they should pursue processors with low energy consumption.

Flexible usage of ODS: Similar to HEFT, LEC, and MR, ODS can be combined with SOEA to form a complete approach to address dynamic DAG scheduling on multiprocessor systems. Since by varying l and θ ODS may generate a set of solutions, each of which could go through SOEA for further optimisation on reliability or energy consumption, it has flexible usage to suit different demands. In the above example, we pick up the solution with the shortest makespan, ignoring energy and reliability. Together with SOEA, we can constrain both the energy cost and reliability and optimise the makespan. For example, a constraint on energy consumption can be implemented in SOEA, and among the set of final solutions (generated by ODS and going through SOEA), the one satisfying the reliability requirement and minimising the makespan is selected. It is also possible to take a multi-objective optimisation perspective. For instance, SOEA ensures that the constraint on reliability is satisfied, and a Pareto front between makespan and energy can be formed from the set of final solutions.

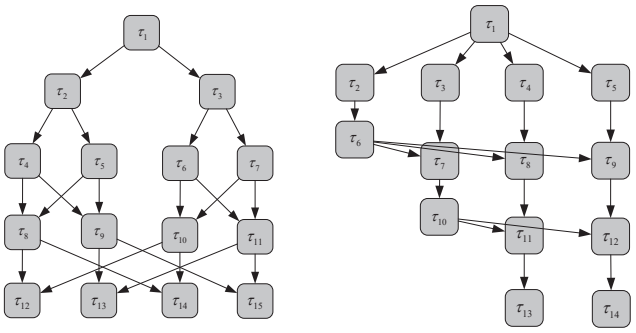
ODS pays a price of iterations and gets a reward of solutions with different strength. Under the fixed demand, it is possible to derive the most suitable l and θ without iterations, which can be investigated in future. In the experiments, we empirically obtain a value of θ and a step size for the search over l.

VI. Experimental Results

In this work, we propose ODS+SOEA to address dynamic DAG scheduling on multiprocessor systems, considering reliability, energy consumption, and makespan. ODS allocates the DAG nodes to processors and SOEA determines their operating frequencies. In this section, we perform comprehensive evaluation in terms of both the approaches for comparison and the diversity of DAG applications.

A. Algorithms for Comparison

We compare our proposed ODS+SOEA with the state-of-the-art approaches LRDSA [5], ESRG [6], as well as MR+SOEA, LEC+SOEA, and HEFT+SOEA. As MR, LEC, and HEFT only perform task allocation, we use SOEA to help them decide the operating frequencies of processors, for a fair comparison. All these existing approaches are heuristics and able to optimise reliability, makespan, as well as energy cost, to some extent, yet with different orientations. LRDSA is a joint makespan and reliability optimisation algorithm. ESRG targets reliability maximisation under energy constraints. MR+SOEA, LEC+SOEA, and HEFT+SOEA focus on reliability, energy cost, and makespan, respectively, as discussed earlier. We would like to make a note that LRDSA has been followed up by a number of more recent studies, however, with different orientations, such as resource utilisation optimisation.



(a) An example FFT DAG application with $\rho = 2$

(b) An example GE DAG application with $\rho = 5$

Fig. 2. Examples of FFT and GE DAG applications.

B. Experimental Platform and DAG Applications

The occurrence probability of task failure is small and hard to control on real systems. Therefore, similar to most existing studies, we run simulations. Using C++, we simulate a fully connected heterogeneous 32-processor system on a laptop with 16 GB of Memory and an Inter core i7 processor. Parameters of the processors are randomly set in the following ranges: $P^* \in [0.4, 0.8]$, $c \in [0.8, 1.3]$, $f \in [0.3, 1.0]$, $\alpha \in [2.7, 3.0]$, $\lambda_F \in [0.1E-5, 1.0E-5]$, $d \in [1, 3]$, which reflect the real-world characteristics, such as for Intel Mobile Pentium III and ARM Cortex-A9. It is to be noted that redundancy, which can be deployed to achieve extremely high reliability, is not considered in this work.

We perform evaluation under *Fast Fourier Transform (FFT) DAG applications*, *Gaussian Elimination (GE) DAG applications*, and *random DAG applications*, which have different characteristics described as follows:

- FFT applications have a high degree of parallelism. Figure 2(a) shows an example with $|N| = 15$ sub-tasks. This number of sub-tasks $|N|$ is computed by
$$\begin{aligned} |N| &= (2 \times 2^{\rho} - 1) + 2^{\rho} \times \log(2^{\rho}) \\ &= (2 + \rho) \times 2^{\rho} - 1, \end{aligned}$$
where ρ is a natural number, i.e., $\rho \in Z^+$ [8].
- GE applications have a low degree of parallelism. Figure 2(b) shows an example with $|N| = 14$ tasks. The number of sub-tasks is computed by $|N| = \big(\rho^2 + \rho - 2\big)/2$, where ρ is a natural number, $\rho \in Z^+$.
- As the name suggests, random applications are randomly generated, with random parallelism and computation/communication cost.

C. Evaluation Metrics

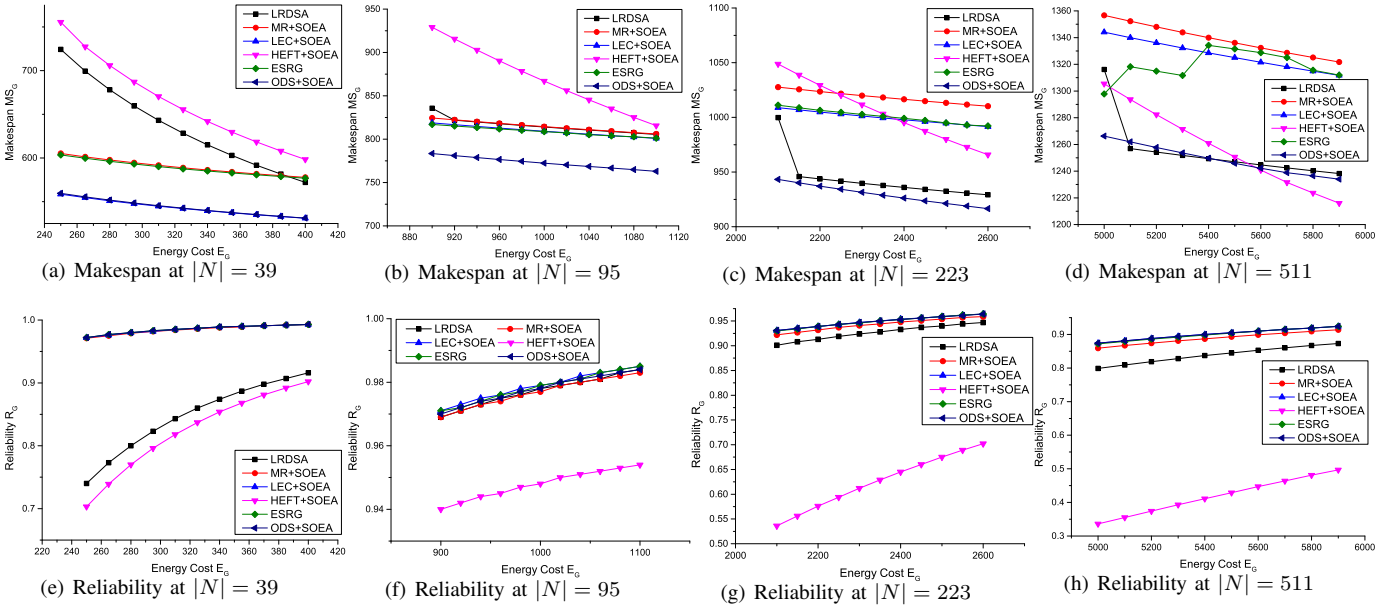
As discussed at the end of Section V, our approach ODS+SOEA can be used flexibly, such as treating reliability and energy cost as objectives or constraints. However, the other existing approaches have limitation in how to be used. For instance, LRDSA assumes that all processors run at the same and stable frequency, the maximal frequency by default, which is normalised to 1. The energy overhead can be adjusted by changing the frequency. It is not able to precisely control the value of reliability, making it impossible to take reliability as a constraint. For fair comparison, the following evaluation scenario is designed. We take the energy consumption as a strict constraint for all the six approaches, and vary it to create different cases. Under the energy constraint, we run the other five approaches and find the best reliability R_{best}. From the solution set generated by ODS+SOEA, the one with the closest reliability to R_{best} is picked up, and then compared to the other five approaches on makespan. Under the same energy constraint, compared to the approaches with similar reliability, we expect to see better makespan to show that our approach is superior. Compared to the approaches with worse reliability, it would be dominating if ours has better makespan, and acceptable if not. The frequency of each processor can be adjusted in the step of 0.0001.

D. Results and Analysis

Our experiments are conducted on three groups of DAG applications: FFT, GE, and random. Each group is further divided into four sub-groups. Each sub-group has 30 different DAGs with the same number of nodes, in the range of 35 to 550. For each sub-group, we report the average performance over the 30 DAGs. The units are omitted without affecting the comparison. The range over the number of nodes in a DAG under evaluation is sufficient to represent most, if not all, embedded applications, such as in the domains of automotive, avionics, industrial automation, and 5G networks, based on our collaboration with the industry partners. DAGs with more than 100 nodes are rarely seen in embedded systems. For example, a real-world DAG from the automotive industry is analysed in [26], with a total of 9 nodes.

Experiment 1: We compare our proposed ODS+SOEA against the five existing approaches, LRDSA, ESRG, MR+SOEA, LEC+SOEA, and HEFT+SOEA, under the FFT DAG applications. As shown in Figure 2 and explained in Section VI-B, the number of sub-tasks and structure of an FFT DAG is controlled by the parameter ρ. We set ρ to be 3, 4, 5, and 6. Correspondingly, the number of sub-tasks $|N|$ is 39, 95, 223, and 511 in the four sub-groups. The computation cost T_{k,τ_i} of the sub-task τ_i and the communication cost $w_{i,j}$ between the sub-tasks τ_i and τ_j are randomly generated [27], both in the range $[10, 100]$. For $|N| = 39$, $|N| = 95$, $|N| = 223$, and $|N| = 511$, the energy constraint is given from 250 to 400 with a step of 20, from 900 to 1100 with a step of 20, from 2100 to 2600 with a step of 50, and from 5000 to 5900 with a step of 100, respectively.

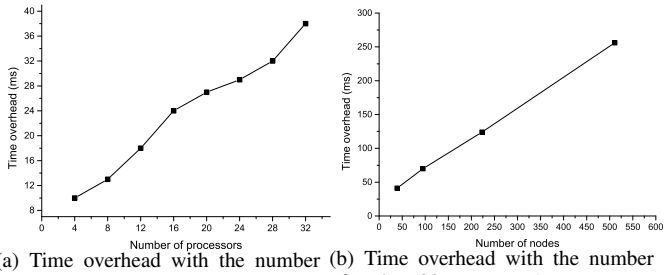
The comparison results are presented in Figure 3, where Figures 3 (a-d) show the makespan MS_G when the number of sub-tasks is 39, 95, and 223, 511, respectively, and Figures 3 (e-h) show the corresponding reliability. It is noted that some algorithms may have similar results and hard to distinguish in the figure. From the first three cases in Figures 3 (a-c) and (e-g), as well as most energy constraints of the fourth case $|N| = 511$ in Figures 3 (d) and (h), we can see that our ODS+SOEA has the highest reliability and shortest



(a) Makespan at $|N| = 39$ (b) Makespan at $|N| = 95$ (c) Makespan at $|N| = 223$ (d) Makespan at $|N| = 511$

(e) Reliability at $|N| = 39$ (f) Reliability at $|N| = 95$ (g) Reliability at $|N| = 223$ (h) Reliability at $|N| = 511$

Fig. 3. Comparison results under FFT DAGs



(a) Time overhead with the number of processors (39 nodes) (b) Time overhead with the number of nodes (32 processors)

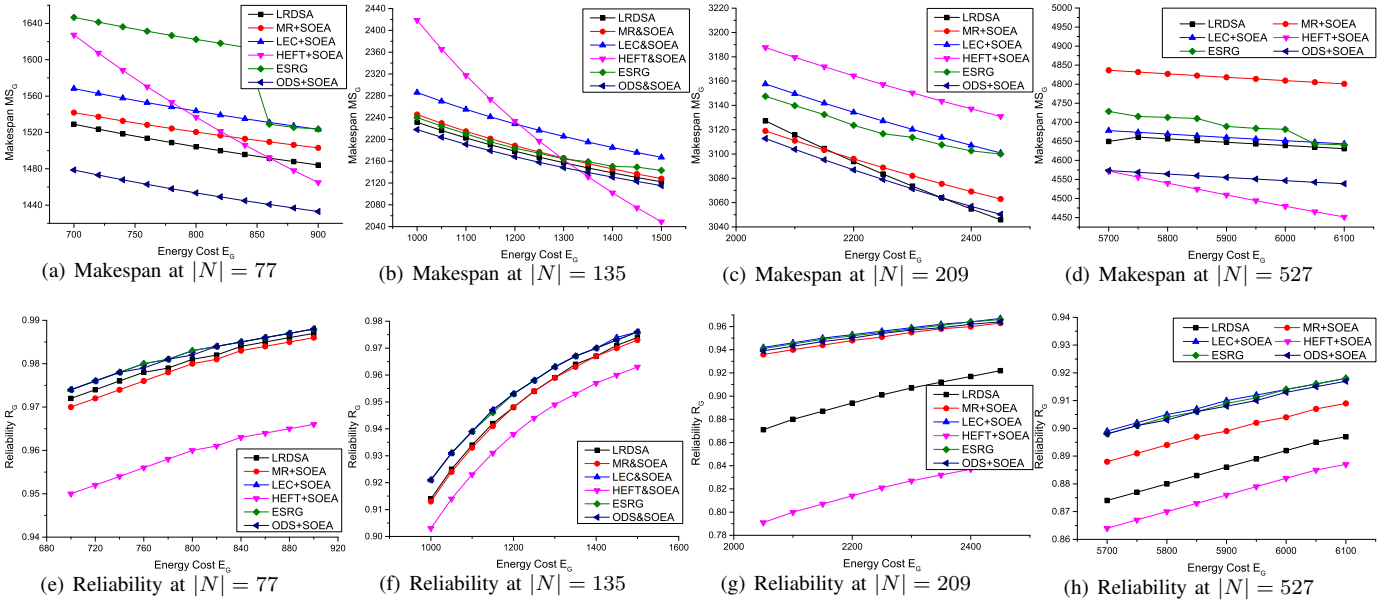
Fig. 4. The runtime overhead of ODS+SOEA

makespan, clearly superior to all the existing approaches. When $|N| = 511$ and the energy constraint is relaxed (i.e., being large), HEFT+SOEA achieves better makespan, yet with significantly lower reliability. Following this observation, HEFT+SOEA may be the best approach if an embedded system is able to tolerate high energy consumption and low reliability. Generally, as energy consumption increases, makespan becomes shorter and reliability gets improved. In addition, reliability deteriorates as the size of DAG increases, indicating that large-scale applications are more likely to incur failure.

The runtime overhead of the proposed ODS+SOEA is reported in Figure 4, where (a) varies the number of processors from 4 to 32 with the number of nodes fixed at 39, and (b) varies the number of nodes at 39, 95, 223, and 511, with the number of processors fixed at 32. The results show that our approach is useful for most practical embedded applications with scalability. Taking a common scenario with 39 nodes and 8 processors as an example, the runtime overhead is less than 15ms. We would like to make a note that the runtime is measured on a general-purpose computer and could be considerably reduced with hardware acceleration when used in practice.

Experiment 2: We compare our proposed ODS+SOEA with the five existing approaches under the GE DAG applications. Similar to Experiment 1, we vary the number of sub-tasks and energy constraints for evaluation. Figure 2 with the explanation in Section VI-B shows that the number of sub-tasks is determined by the parameter ρ. In this experiment, we set ρ to be 12, 16, 20, and 32. Correspondingly, the number of sub-tasks $|N|$ is 77, 135, 209, and 527 in the four sub-groups. The computation and communication cost for each sub-task are again randomly generated, both in the range $[10, 100]$. For $|N| = 77$, $|N| = 135$, $|N| = 209$, and $|N| = 527$, the energy constraint is given from 700 to 900 with a step of 20, from 1000 to 1500 with a step of 50, from 2100 to 2500 with a step of 50, and from 5700 to 6100 with a step of 50, respectively.

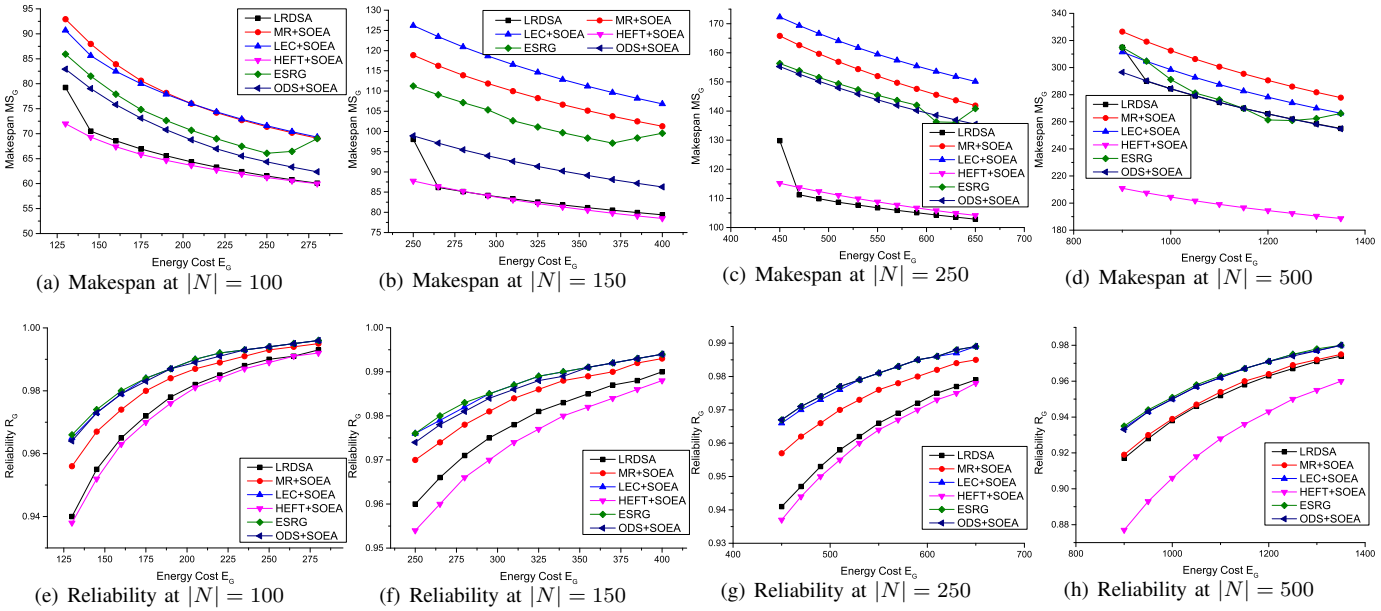
The comparison results are presented in Figure 5, where Figures 5 (a-d) show the makespan MS_G when the number of sub-tasks is 77, 135, 209, and 527, respectively, and Figures 5 (e-h) show the corresponding reliability. From the two cases in Figures 5 (a) (c) and (e) (g), as well as most energy constraints of the case $|N| = 135$ in Figures 5 (b) and (f), we can see that our ODS+SOEA has the highest reliability and shortest makespan, better than all the existing approaches. When $|N| = 135$ and the energy constraint is large, HEFT+SOEA achieves shorter makespan, yet with lower reliability. Similarly, when $|N| = 527$, HEFT+SOEA has shorter makespan and lower reliability. In general, HEFT+SOEA performs poorly on reliability, which clearly shows the limitation of HEFT, especially considering that the result has been optimised by SOEA. In addition, HEFT+SOEA loses its makespan when the energy constraint is strict. The main reason is that HEFT does not take energy cost and reliability into account, which stresses the importance to treat makespan, reliability, and energy together in DAG scheduling.



(a) Makespan at $|N| = 77$ (b) Makespan at $|N| = 135$ (c) Makespan at $|N| = 209$ (d) Makespan at $|N| = 527$

(e) Reliability at $|N| = 77$ (f) Reliability at $|N| = 135$ (g) Reliability at $|N| = 209$ (h) Reliability at $|N| = 527$

Fig. 5. Comparison results under GE DAGs



(a) Makespan at $|N| = 100$ (b) Makespan at $|N| = 150$ (c) Makespan at $|N| = 250$ (d) Makespan at $|N| = 500$

(e) Reliability at $|N| = 100$ (f) Reliability at $|N| = 150$ (g) Reliability at $|N| = 250$ (h) Reliability at $|N| = 500$

Fig. 6. Comparison results under random DAGs

Experiment 3: We compare our proposed ODS+SOEA with the five existing approaches, LRDSA, ESRG, MR+SOEA, LEC+SOEA, and HEFT+SOEA, under random DAG applications, whose generation is controlled by six parameters, i.e., the number of sub-tasks, shape, average computation cost, communication to computation ratio, the number of processors, and the heterogeneity factor. The parameter "shape" affects the height and width of a DAG. The heterogeneity factor, which is defined in the interval of $[0.1, 1]$, reflects the consistency between sub-tasks. The communication to computation ratio represents the execution of the sub-tasks relative to the amount of data transmitted between them. A small ratio indicates that the generated DAG is computationally intensive, otherwise communicatively intensive. In this experiment, the parameters shape, average computation cost, communication to computation ratio, heterogeneity factor, and number of processors are set to be 2, 10, 1, 0.8, and 32, respectively. Similar to Experiments 1 and 2, a total of 4 sub-groups are generated, each of them consisting of 30 DAGs. The numbers of sub-tasks in each DAG for the four sub-groups are 100, 150, 250, and 500, respectively. The corresponding energy constraints are given from 125 to 275 with a step of 25, from 250 to 400 with a step of 25, from 450 to 650 with a step of 25, and from 900 to 1300 with a step of 50, respectively.

The comparison results are presented in Figure 6, where Figures 6 (a-d) show the makespan MS_G when the number of sub-tasks is 50, 100, 150, and 200, respectively, and Figures 6 (e-h) show the corresponding reliability. From the three cases in Figures 6 (a-c) and (e-g), our ODS+SOEA has

the highest reliability and shorter makespan than MR+SOEA, LEC+SOEA, and ESRG. LRDSA and HEFT+SOEA achieve shorter makespan than ODS+SOEA, with the price of lower reliability. Similarly, in the fourth case of Figure 6 (d) and (h), HEFT+SOEA achieves shorter makespan with lower reliability. Therefore, ODS+SOEA is in the same position as HEFT+SOEA in the Pareto sense. Neither of them dominates the other.

Summary: We have conducted comprehensive evaluation on our proposed approach ODS+SOEA against five state-of-the-art algorithms under three groups of DAGs with high, low, and random parallelism, respectively. Generally, ODS+SOEA performs better than the five existing approaches. In certain cases, HEFT+SOEA achieves shorter makespan with lower reliability. It is to be noted that we have only picked up one solution from the set generated by ODS+SOEA. It is possible that there exists another solution dominating HEFT+SOEA in some of the cases. We would like to state that we do not claim ODS+SOEA to be the dominantly best approach in all cases.

VII. Concluding Remarks

In this work, we have tried to address the dynamic DAG scheduling problem on multiprocessor systems, considering reliability, energy consumption, and makespan. Assuming that the allocation of DAG nodes to processors is given, we propose the first optimal methods, OEA and SOEA, to minimise the energy consumption whilst satisfying the reliability requirement. OEA has a closed-form solution for homogeneous architecture, and SOEA is an algorithm built upon binary search for heterogeneous systems. OEA and SOEA can be used to maximise reliability and respect the constraint on energy consumption as well. SOEA is able to be combined with any scheduling algorithm that allocates DAG nodes to processors.

We present a novel scheduling algorithm ODS that allocates the DAG nodes according to their out-degrees, and considering processors' energy consumption, dynamic finish time of sub-tasks, as well as reliability of sub-tasks on processors. Essentially, we try to allocate those nodes with larger numbers of successors to the processors with early finish time. ODS dominates the widely applied HEFT as it is always able to generate one solution equivalent to HEFT. Combining SOEA with ODS makes a complete solution to dynamic DAG scheduling on multiprocessor systems that can be flexibly used, and performs generally better than the state-of-the-art approaches, including LRDSA, ESRG, MR+SOEA, LEC+SOEA, and HEFT+SOEA. Both SOEA and ODS have polynomial time complexity.

In this work, we decompose the problem to two stages, as in one stage, the optimal solution can be computed. It is possible to solve the problem as a whole, which could lead to better results on one hand, and makes it very challenging to devise effective approaches that can achieve these better results on the other hand, due to the complexity. This is a promising future research direction to pursue. In addition, ODS may be further improved. First, as a heuristic, it may get better performance if more factors, such as the computation and communication cost, are considered. Second, it is now iterated over l and θ. The time complexity can be further reduced if there are methods to determine l and θ according to specific demands, as has been discussed when summarising ODS.

Besides, this work can be extended towards hard timing constraints, especially when considering multiple preemptive (and even migrating) DAGs. Currently, we deal with a single DAG, and can trivially support multiple non-preemptive DAGs. DAGs could have mixed criticality levels, as derived from the practical context, and may have different priorities, which, e.g., could be assigned according to deadlines. The existing literature on DAG scheduling towards timing guarantees is still quite limited, even if the other objectives like energy consumption and reliability are not taken into account. Generally, the scheduling methods are not able to exploit the parallelism offered in multiprocessor systems, and the analyses, although safe, are rather conservative and unsuitable for practical design.

Acknowledgment

Jing Huang was supported by the Natural Science Foundation of China Grant No. 61902118, 61932010, China Postdoctoral Science Foundation No. 2019M662771.

References

[1] S. Saidi, S. Steinhorst, A. Hamann, D. Ziegenbein, and M. Wolf, "Special session: Future automotive systems design: Research challenges and opportunities," in *CODES+ISSS*, 2019.

[2] D. Ziegenbein, S. Saidi, X. S. Hu, and S. Steinhorst, "Future automotive HW/SW platform design (Dagstuhl Seminar 19502)," *Dagstuhl Reports*, vol. 9, no. 12, pp. 28–66, 2020. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2020/12010

[3] AUTOSAR, "Adaptive platform 19.03," 2019. [Online]. Available: https://www.autosar.org/standards/adaptive-platform/adaptive-platform-1903/

[4] B. Zhao, H. Aydin, and D. Zhu, "On maximizing reliability of real-time embedded applications under hard energy constraint," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 3, pp. 316–328, Aug 2010.

[5] X. Tang, K. Li, M. Qiu, and E. H.-M. Sha, "A hierarchical reliability-driven scheduling algorithm in grid systems," *Journal of Parallel and Distributed Computing*, vol. 72, no. 4, pp. 525–535, 2012.

[6] G. Xie, Y. Chen, X. Xiao, C. Xu, R. Li, and K. Li, "Energy-efficient fault-tolerant scheduling of reliable parallel applications on heterogeneous distributed embedded systems," *IEEE Transactions on Sustainable Computing*, vol. 3, no. 3, pp. 167–181, July 2018.

[7] J. Zhou, J. Sun, X. Zhou, T. Wei, M. Chen, S. Hu, and X. S. Hu, "Resource management for improving soft-error and lifetime reliability of real-time mpsocs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 12, pp. 2215–2228, 2018.

[8] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.

[9] G. Xie, Z. Gang, R. Li, and K. Li, "Energy-aware processor merging algorithms for deadline constrained parallel applications in heterogeneous cloud computing," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 62–75, 2017.

[10] W. Chang, A. Probstl, D. Goswami, M. Zamani, and S. Chakraborty, "Battery- and aging-aware embedded control systems for electric vehicles," in *2014 IEEE Real-Time Systems Symposium*, Dec 2014, pp. 238–248.

[11] Y. Ma, J. Zhou, T. Chantem, R. P. Dick, S. Wang, and X. S. Hu, "Online resource management for improving reliability of real-time systems on big-little type mpsocs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 1, pp. 88–100, Jan 2020.

[12] A. Dogan and F. Ozguner, "Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 308–323, March 2002.

[13] L. Zhao, Y. Ren, and K. Sakurai, "Reliable workflow scheduling with less resource redundancy," *Parallel Computing*, vol. 39, no. 10, pp. 567–585, 2013.

[14] A. Benoit, M. Hakem, and Y. Robert, "Fault tolerant scheduling of precedence task graphs on heterogeneous platforms," in *2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 2008, pp. 1–8.
[15] S. M. Shatz and J.-P. Wang, "Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems," *IEEE Transactions on Reliability*, vol. 38, no. 1, pp. 16–27, 1989.
[16] A. Girault and H. Kalla, "A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate," *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 4, pp. 241–254, 2008.
[17] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, and K. Li, "Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster," *Information Sciences*, vol. 319, pp. 113–131, 2015.
[18] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 8, pp. 1374–1381, 2010.
[19] Q. Qiu and M. Pedram, "Dynamic power management based on continuous-time markov decision processes," in *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, 1999, pp. 555–561.
[20] J. Huang, R. Li, J. An, D. Ntalasha, F. Yang, and K. Li, "Energy-efficient resource utilization for heterogeneous embedded computing systems," *IEEE Transactions on Computers*, vol. 66, no. 9, pp. 1518–1531, 2017.
[21] G. Xie, G. Zeng, Y. Chen, Y. Bai, Z. Zhou, R. Li, and K. Li, "Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems," *IEEE Transactions on Services Computing*, 2017.
[22] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, and N. Guan, "Energy-efficient real-time scheduling of dags on clustered multi-core platforms," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019, pp. 156–168.
[23] A. Melani, M. Bertogna, V. Bonifaci, A. Marchettispaccamela, and G. Buttazzo, "Response-time analysis of conditional dag tasks in multiprocessor systems," pp. 211–221, 2015.
[24] J. Fonseca, G. Nelissen, and V. Nelis, "Schedulability analysis of dag tasks with arbitrary deadlines under global fixed-priority scheduling," *Real-time Systems*, vol. 55, no. 2, pp. 387–432, 2019.
[25] M. Taufer and A. L. Rosenberg, "Scheduling dag-based workflows on single cloud instances: High-performance and cost effectiveness with a static scheduler," *The International Journal of High Performance Computing Applications*, vol. 31, no. 1, pp. 19–31, 2017.
[26] M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, "Latency-aware generation of single-rate dags from multi-rate task sets," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 226–238.
[27] https://sourceforge.net/projects/taskgraphgen/, online, (2015).